Jaatun *et al. Journal of Cloud Computing: Advances, Systems and Applications* 2012, **1**:13
http://www.journalofcloudcomputing.com/content/1/1/13

Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH Open Access

The design of a redundant array of independent net-storages for improved confidentiality in cloud computing

Martin Gilje Jaatun[1*], Gansen Zhao[2], Athanasios V Vasilakos[3], Åsmund Ahlmann Nyre[1], Stian Alapnes[4] and Yong Tang[2]

Abstract

This article describes how a Redundant Array of Independent Net-storages (RAIN) can be deployed for confidentiality control in Cloud Computing. The RAIN approach splits data into segments and distributes segments between multiple storage providers; by keeping the distribution of segments and the relationships between the distributed segments private, the original data cannot be re-assembled by an observer. As long as each segment is small enough, an individual segment discloses no meaningful information to others, and hence RAIN is able to ensure the confidentiality of data stored in the clouds. We describe the inter-cloud communication protocol, and present a formal model, security analysis, and simulation results.

1 Introduction

Security concerns are frequently cited [1,2] as one of the major obstacles to cloud computing adoption. In a traditional outsourcing scenario, technical and organizational security mechanisms contribute to protect a customer's data, but the most important factor is that the customer establishes a trust relationship with the provider. This implies that the customer acknowledges that if the provider is evil, the customer's data may be used improperly [3].

One aspect of Cloud Computing can be described as "outsourcing on steroids"; where both storage and processing is handled by one or several external providers, and where the provider(s) may be in a different jurisdiction than the customer. Not knowing where your data is physically located may be uncomfortable to the customer, and personal data may even be illegal to export from some jurisdictions [4]. Just like with traditional offshoring, settling disputes is more challenging when the provider may be on a different continent, which is all the more reason to limit the degree to which the customer has to trust the provider. This is the "need to know" principle in a nutshell - if the provider does not need to read the information, why should it be allowed to?

In this article, we explore a Cloud Computing scenario where the dependency on *trust* will be reduced through a divide-and-conquer approach, where each actor gets access to sufficiently small units of data so as to minimize confidentiality concerns. In a way, our approach is the opposite of the aggregation problem in database security [5] – we *de-aggregate* the sensitive data.

The remainder of the article is structured as follows: In Section 2 we outline the background for our contribution. In Section 3 we sketch our solution, and detail the protocol between the various actors further in Section 4. We present a formal model in Section 5, and provide a security analysis in Section 6. We discuss implementation considerations in Section 4.6 and present simulation results in Section 7. We discuss our contribution in Section 8, outline further work in Section 9, and offer our conclusions in Section 10.

2 Background

Cloud computing provides on-demand services delivered via the Internet, and has many positive characteristics such as convenience, rapid deployment, cost-efficiency,

*Correspondence: martin.g.jaatun@sintef.no
[1] SINTEF ICT, Trondheim, Norway
Full list of author information is available at the end of the article

and so on. However, we have shown [6] that such off-premises services cause clients to be worried about the confidentiality, integrity and availability of their data.

In previous work [7], we identified five deployment models of cloud services designed to ease users' security concerns:

- **The Separation Model** separates storage of data from processing of data, at different providers.
- **The Availability Model** ensures that there are at least two providers for each of the data storage and processing tasks, and defines a replication service to ensure that the data stored at the various storage providers remains consistent at all times.
- **The Migration Model** defines a cloud data migration service to migrate data from on storage provider to another.
- **The Tunnel Model** defines a data tunneling service between a data processing service and a data storage service, introducing a layer of separation where a data processing service is oblivious of the location (or even identity) of a data storage service.
- **The Cryptography Model** extends the tunnel model by encrypting the content to be sent to the storage provider, thus ensuring that the stored data is not intelligible to the storage provider.

By use of these deployment models, we have shown [1] that through duplication and separation of duty, we can alleviate availability and integrity concerns, and to some extent also confidentiality by implementing encrypted storage. However, even with encrypted storage, we still have to trust the encryption provider with *all* our data. Furthermore, if the data needs to be processed in the cloud, the cloud processing provider in general also needs to have access.

The main motivation for confidentiality control in the cloud is currently various privacy-related legislation forbidding the export of sensitive data out of a given jurisdiction, e.g. the Privacy legislation in the EU [4]. The current solution to this problem has been to sidestep it: By offering geolocalized cloud services, where a customer may request the cloud provider to ensure that the sensitive data is only stored and processed on systems that are physically located in a geographically defined area, e.g., within the borders of the European Union. However, this is rapidly becoming a moot point, since cloud service providers typically run global operations, and although data might physically reside in one jurisdiction, it will in principle be accessible from anywhere in the world.

Although misappropriation of data by cloud providers has not been documented, Jensen et al. [8] show that current cloud implementations may be vulnerable to attack, and the first examples of Cloud compromises have surfaced [9]. Ristenpart et al. [10] demonstrate that even supposedly secret information such as where a given virtual machine is running may be inferred by an attacker, highlighting another attack path. Furthermore, insider malfeasors can be a challenge for any organization, and an incident at Google shows they are as vulnerable as anyone [11].

Krautheim [12] proposes to achieve cloud security through the introduction of Trusted Platform Modules (TPM) in all datacenter equipment. It is not clear, however, how the user could verify that a TPM is indeed present in any given cloud infrastructure. You might argue that the cloud provider could assert, and have an auditor confirm that they are using a TPM, but this is really not much better than today's situation where providers are asserting that they will treat your data properly, and all their certifications is a testament to them staying true to their words.

2.1 Previous work on security through splitting data

The Free Haven project [13] describes a collaborative distributed storage system, where participants are allowed to store (or publish) data by offering to store data for others, in the same general fashion of peer-to-peer file sharing. The Free Haven project does not provide a new solution for the anonymous communications channel, but uses a set of anonymous remailers as a basis. The Free Haven project makes no assumptions on the participants being honest, but uses a reputation system to identify non-cooperative (or dishonest) nodes.

The OceanStore [14] system is also based on distributed storage, but is not concerned with ensuring anonymity of the individual users.

The ShareMind framework [15,16] offers distributed privacy-preserving[a] computations, based on the principles of secure multiparty computations. Sharemind is not focused on (anonymous) storage; the current prototype solution is based on distributing data from one source among three nodes referred to as *data miners*, and is only secure as long as the three miners do not collude.

2.2 A brief introduction to Botnets

A *botnet* is a collection of compromised computers (*bots*) which are controlled by a human *botmaster*, often through a convoluted hierarchy of subnodes to evade detection and disclosure of the network and its owner. This is illustrated in Figure 1, inspired by by Wang et al. [17].

A traditional C&C (Command & Control) botnet is created by infecting regular PCs with malware that opens up a backdoor. Furthermore, the infected hosts actively poll a shared communication medium (typically: An Internet Relay Chat channel) for instructions. When correctly tagged instructions are observed on the shared medium,

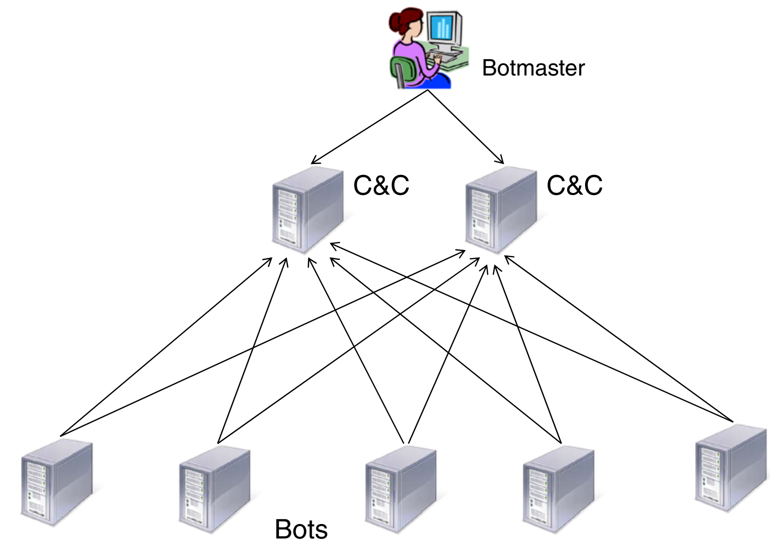


Figure 1 A traditional C&C botnet.

the infected host obeys without verifying (or even knowing) the identity of the issuer.

Botnets are used for mundane tasks such as sending unsolicited commercial email (spam) and performing Distributed Denial of Service (DDoS) attacks, but could in theory be used for any task that is amenable to distributed computing on nodes with modest processing and memory resources. In the following, we will show that we can exploit some properties of a botnet, not by infecting Cloud nodes, but by running autonomous agents as legitimate processes on Cloud processing providers.

3 Approach

We have extended the deployment models [1] with a new concept where data is split up and kept by several independent (non-colluding) storage providers in a Redundant Array of Independent Net-storages (RAIN) [6], in such a manner that a single chunk does not compromise confidentiality [18]. The data can then be stored using one or several cloud storage providers (duplicated, according to the deployment models).

3.1 Assumptions

The RAIN solution makes the following security assumptions:

1. We have a file (or dataset) that has been divided into small chunks
2. A provider will not be able to link two different chunks of the same dataset, should it gain access to them
3. The cloud service providers can be classified as “Honest but curious”, i.e., we expect them to carry out the protocol faithfully, but they may try to access the information either through collusion or other means.
4. There are enough simultaneous users to make anonymity feasible[b]
5. We have a lightweight authentication mechanism which can be used to regulate access to a data item
6. The C&C node has a list of “honest” Cloud Processing providers, and their public keys
7. The C&C node maintains a record of all data IDs
8. The C&C node maintains a record of all nonces generated by legitimate users and itself, for as long as a data ID is active
9. The user maintains a log of any outstanding requests sent to the C&C node, and will reject any unsolicited responses
10. The C&C node maintains a log of any outstanding requests sent to cloud processing providers, and will reject any unsolicited responses

The adversarial model is less powerful than Dolev-Yao, in that we assume that an adversary can observe all traffic, and possibly insert traffic, but not in general *delete* traffic (e.g. does not carry the message).

3.2 Using Botnets for non-nefarious purposes

We propose to organize the various elements in our distributed cloud architecture as a traditional multi-tier Command & Control (C&C) botnet, e.g. as described by Wang et al. [17].

We implement the shared medium as a cloud multicast service which can be freely accessed by anyone. To prevent tracing, we employ an approach similar to Onion Routing [19] when issuing commands from the C&C node.

We introduce a new type of cloud service provider which assumes the role of the botnet C&C node, and which is in charge of assembling the information and presenting it. This keeps all processing in the cloud, but leaves us with the problem that we have to trust this provider with our information. The resulting configuration is conceptually illustrated in Figure 2a.

The key property of the solution is that all the subnodes (cloud processing providers) and leaf nodes (cloud storage providers) only get to observe a small subset of a user's data, and that these nodes are prevented from associating a given piece of data with a specific user, or with other pieces of the same dataset. Ultimately, it will be like breaking open a large number of jigsaw-puzzles and distributing the pieces among storage providers – a single provider will not be able to determine where the pieces come from, or even if they are part of the same puzzle. Note that we do not propose to make the cloud processing provider work with encrypted data; the confidentiality control is achieved through the de-aggregation of information, and hiding the relationships between the processing providers. Also note that assuming the volume of such "botnet computations" is large enough (i.e., many enough users employ this technique), it is also possible to re-use providers, since it will not be possible for a provider to relate two different processing tasks with each other. Note that requiring a certain number of users is not unusual in similar applications, e.g., the TOR designers [19] make the same assumption regarding TOR's ability to provide privacy for its users.

For the truly paranoid, it could be possible to introduce uncertainties by routinely accessing bogus data, but although the user will know which data is real, and which is bogus, this will introduce the need for some "intelligence" on the client (to separate the wheat from the chaff), and we find ourselves transported to the alternative solution presented in Section 3.8.

3.3 A revival for autonomous mobile agents

Mobile agents [20] have already been suggested as a viable paradigm for services in Cloud Computing [21-23]. Many traditional security concerns with mobile agents are related to security of the host platform [24], but in a cloud setting these are mostly alleviated through the virtualization strategies employed. As will be detailed below, the mobile agent paradigm is a very good fit for the intermediate nodes in our design.

3.4 Example 1 – digital image

To illustrate the concept, we will in the following consider the storing of bitmap images in the cloud. Figure 3a shows a 480X480 image. The image is sliced into a 10X5 grid, of 48X98 pixels each. For our purposes, a single slice of the picture does not reveal much useful information to the observer, and this information can be stored unencrypted as long as it is not possible to combine it with the other slices.

The C&C node performs the slicing of the image, and randomly distributes the slices among (say) 10 subnodes. Each subnode then stores the slices independently

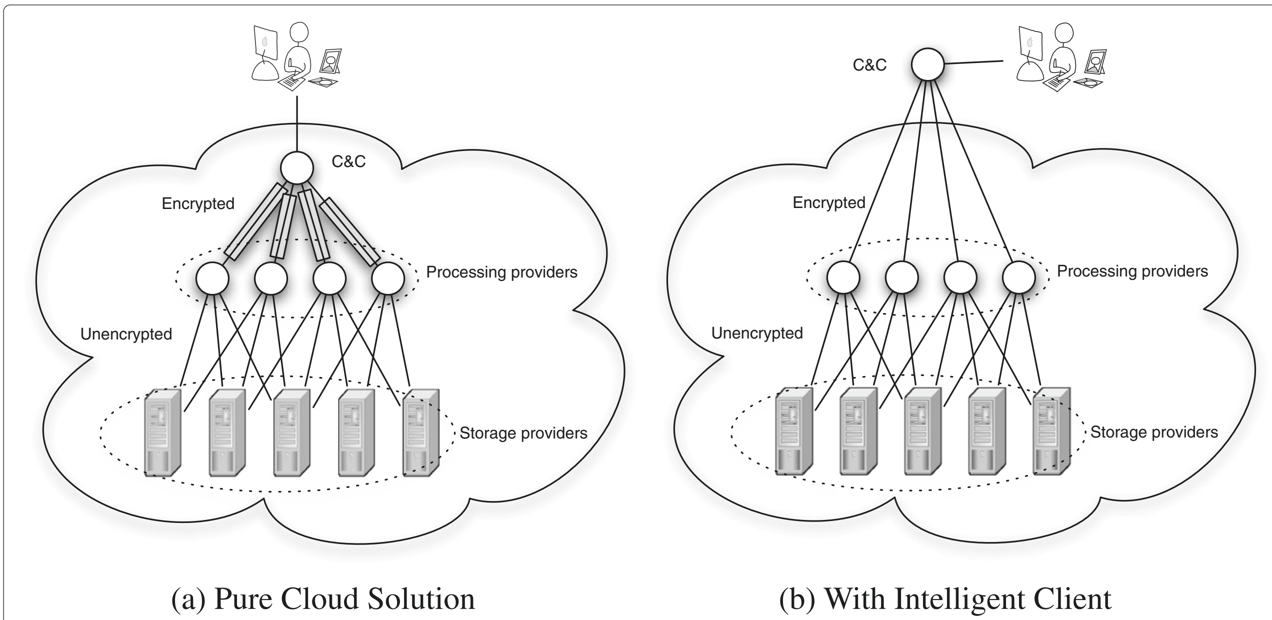
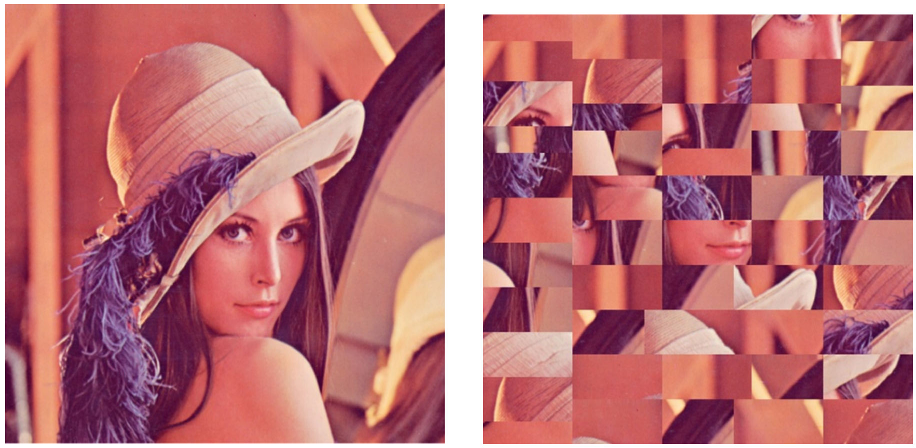


Figure 2 Divide-and-conquer Cloud Security. (**a**) Pure Cloud Solution. (**b**) With Intelligent Client .



(a) Original Lena Image (b) Randomly Segmented Lena Image

Figure 3 Segmentation and Randonmization. (**a**) Original Lena Image. (**b**) Randomly Segmented Lena Image.

using as many cloud storage providers as available (ideally one for each slice, but even for this small example we would probably be hard pressed to find 50 independent providers). To prevent observability, the subnodes may use an encrypted tunnel to transfer the data to the storage providers.

It is the responsibility of the C&C node to keep track of which subnode has received which slices, but also to record the location. When the image is to be retrieved, the user will instruct the C&C node to fetch all the slices.

Admittedly, this is a toy example, with all the real processing being performed by the C&C node – the real challenge comes when it is required to perform complicated processing on each subnode. This will also introduce the need for more sophisticated "slicing" of data.

Note that, without the proper knowledge of the distribution of the slices, it is very unlikely that the original image can be reconstructed. Given all the slices, it is still not easy to reconstruct the original image, if the number of slices is large enough. Figure 3b is an example of the reconstructed image without knowledge of the slices' order. The reconstructed image does not look much like the original image. If the image is sliced into even smaller slices, the reconstructed image would be even more different from the original image.

3.5 Example 2 – electronic document

The need to permutate the different slices becomes more evident if we consider the example of an online document stored in the Cloud. The naïve solution might be to let every N character be stored at each cloud storage provider, but if the number of providers is as low as three, the risk of a single provider inferring the missing characters is too high for our liking. However, if there is a sufficiently high number of users in total, and the cloud provides cannot differentiate between the users, even this solution may provide adequate security.

Updating a stored document will currently require storing everything anew as if it were a new document, since there are no relationships between the various slices. A possible future extension could be to allow the C&C node to keep track of changes, and only add new data. Note that although the previous example used contiguous image regions as slices (see Figure 3b), this is merely for illustration; both a document and an image could be sliced, e.g., by selecting every n^{th} byte.

3.6 Criteria

Since we perform the slicing and distribution of data in order to achieve data confidentiality, it is important that the slicing and distribution processes adhere to the following criteria:

- Data must be sliced into segments small enough such that each segment bears no meaningful information to malicious entities. With data sliced in this way, malicious entities may be able to access an individual data segment, but the access to the data segment should not compromise the confidentiality of the data as a whole.
- Data segments must be distributed in a random manner, such that it is not possible to establish the relationships between data segments without knowledge of the original data. The relationships between data segments are kept secret by the data owner.

With the above two criteria strictly enforced, the proposed approach would be able to ensure the confidentiality of data. This is achieved without encrypting the data.

3.7 Minimal data unit for processing

In the presented example, the minimal data unit that can be handled by a cloud processing provider is the entire image (e.g. for image manipulation). A similar situation is reasonable to expect for documents, whereas database operations may only need access to a limited number of records, not the whole database.

3.8 Alternative solution

If we are unwilling to trust the C&C provider described in the previous section, an alternative solution is place this functionality on the client, i.e. running on the user's own infrastructure, as figure 2b.

This solution requires a certain amount of computing power present on the client side in order to (re-)assemble all the different pieces of information produced by the cloud processing provides, and may thus not necessarily be considered a pure cloud solution.

4 Protocol

In the following we describe the protocol [25] to be used in RAIN in greater detail. The scheme is dependent on creating a small mix-net [26] in the cloud, and creating a collection of autonomous cloud processing agents explained below (see Figure 4). The autonomous agents will retain a one-to-one relationship with a cloud storage provider. Each agent will have a unique ID that is known to the C&C node, but this knowledge should in principle not allow the C&C node to locate the agent. Furthermore, we need to create a cloud service that can serve as a broadcast medium similar to an IRC channel; for simplicity we will call this the IRC node.

4.1 Protocol for storing data

First, we will describe briefly the main thrust of the protocol, with more details in the following subsections.

Let D be a piece of data to be split and stored in the clouds. The user U will send D to the C&C node, encrypted with that node's public key:

$$U \rightarrow C\&C : \{store{-}full, auth, ID_D, D\}_{K_{C\&C}}$$

Here, "auth" is an authentication token used to verify the user's rights to the dataset[c]. If this message is replayed, it will be ignored; the only way to re-use a data ID is to first delete the data-set that uses it.

The C&C node performs the split [18] such that H can be represented as $H =< D_s, R_s >$ where

- $D_s = \{d_i | i = 1, ..., n\}$
- $R_s = \{< d_i, d_{i+1} > | i = 1, ..., n-1\}$.

Here, R_s specifies how the segments are related to each other; this knowledge is necessary for reassembly. The reulting sequence H can thus be written $H = (d_1, d_2, ..., d_n)$. We need to assign a unique ID (or pseudonym) to each data item, which we in the following refer to as ID_{di}.

The C&C node then distributes H among the cloud storage providers by assigning their respective identifiers

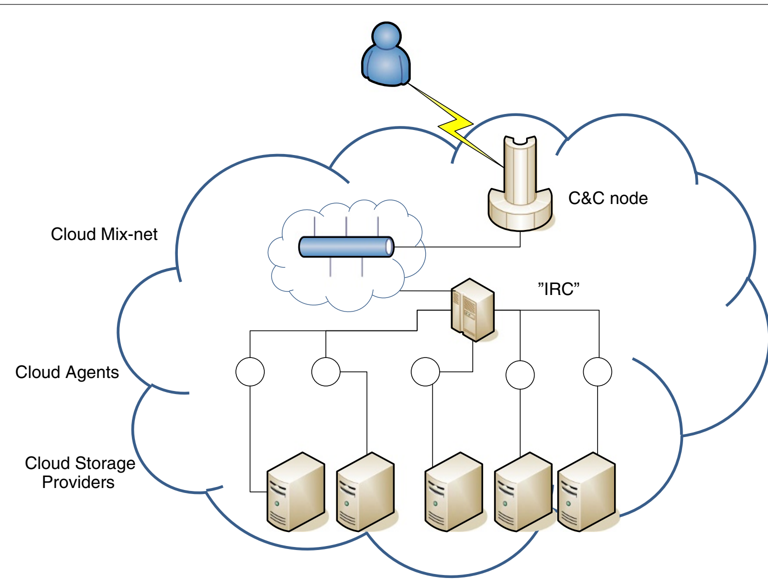


Figure 4 Sketch of solution for storing information.

$(CS_x, \ldots, CS_y)$ to the corresponding d_i, this through the mix-net (see section 4.2 for details) to IRC:

$$\forall i | C\&C \rightarrow IRC : mix(\{CS_j, store\text{–}part, auth, ID_{di}, d_i\})$$

The C&C node needs to maintain a table mapping which data items have been sent to which cloud storage service. Furthermore, it is important that the pseudonyms are unique within a given C&C provider, but created in such a manner that it cannot be determined that two different pseudonyms refer to data items from the same file. As mentioned in Section 3.1, we're assuming that the traffic volume will contribute to hide which items belong to which datasets, so although we are effectively broadcasting the mapping tables, this should not matter: An adversary can tell that the data item with pseudonym X is stored with cloud storage service Y, but this information is of little use if there is no way to tie the pseudonym to a dataset (or user). Furthermore, due to the use of the mix-net, the identity of the cloud storage provider is effectively a pseudonym as well.

The IRC node then publishes this information on its broadcast medium, where all the autonomous agents are listening. When an autonomous agent sees its own ID, it copies the associated d_i and stores this at its associated cloud storage provider.

4.2 Mix-net in the cloud

Chaum's original mix-net idea [26] has been employed with some success in the TOR network [19,27]. In the following, we will describe a simplified scheme tailored to the task at hand.

We assume we have a set of n autonomous agents $m_1, m_2, \ldots, m_n$. The agents are running as cloud web services, and their addresses and public keys are known to the C&C node.

By a slight paraphrasing of Chaum [26] we have that when using a single mixer node m_1, communication from Bob to Alice via m_1 occurs as follows:

$$B \rightarrow m_1 : \{R_m, m_1, A, \{R_a, A, store, d_i, CS_j\}_{K_A}\}_{K_m 1}$$

$$m_1 \rightarrow A : \{R_a, A, store, d_i, CS_j\}_{K_A}$$

Here, R_a and R_m are nonces that are discarded upon decryption, and d_i is the item of data to be sent. The only purpose of the nonce here is to prevent repeated sending of identical plaintexts from generating the same ciphertext. The parameter CS_j identifies the storage agent, which is effectively a pseudonym for the Cloud storage provider below. By recursively applying the scheme above, it is possible to extend it to an arbitrary number of mixer nodes.

In our case, the sender is the C&C node, and the final recipient is always the IRC node. The sole purpose of the mix-net is thus to hide the identity of the C&C node from the IRC node. Naturally, this only makes sense if there are multiple C&C nodes in the system as a whole. In the protocol descriptions, we will use the notation $mix(\ldots)$ to indicate that a message is sent through the mix-net.

4.3 IRC node

The IRC node receives a large amount of data items from multiple C&C nodes, and for each data item, the parameter CS_j identifies which *storage agent* should handle the item. The CRC then simply sends the following to *all* storage agents, using a fixed multicast address *Rain*:

$$\forall i | IRC \rightarrow Rain : CS_j, store{-}irc, auth, ID_{di}, d_i$$

The multicast traffic is UDP-based, and there is thus no acknowledgment or retransmission at the transport level. Although not explicitly shown here, an important feature is then that each data item must be sent to *multiple* storage agents; this redundancy both ensures duplication of storage, and introduces error tolerance in case of bit errors in the transmission.

4.3.1 Storage agent

Each storage agent subscribes to the *Rain* multicast address, and thus receives all the data items, but discards all messages that are not addressed to it. Note that the Storage agent ID (CS_j) can be viewed as a pseudonym, since it is never used as a return address in any way, and can thus not be directly linked to the storage agent.

The storage agents need to maintain a record of data items and associated IDs; the IDs are not actually revealed to the cloud storage providers. However, the storage agents are not anonymous to the cloud storage providers, i.e., the cloud storage providers can log the real addresses of the storage agents, but they do not know their pseudonyms.

4.4 Data retrieval

The user may ask the C&C to retrieve a dataset:

$$U \rightarrow C\&C : \{retrieve{-}full, auth, ID_D\}_{K_{C\&C}}$$

When asked to retrieve a dataset, the C&C node will need to ask each storage service via the IRC to return their respective data items:

$$\forall i | C\&C \rightarrow IRC : mix(\{CS_j, retrieve{-}part, ID_{di}\})$$

Note that we do not need to authenticate when retrieving individual data items in order to fulfill any security claims made by RAIN.

Unfortunately, simply running the storage process in reverse by asking for the data does not work, since an observer then quickly could make the link between storage agent pseudonym and its address. Instead, when we need to retrieve a data set, the C&C will instruct the IRC node to issue a "call for data items", listing the IDs of the data items. In order to complicate traffic analysis, the IRC node will also ask for some[d] bogus IDs; these will simply be discarded.

The storage agents that find matching data item IDs in their records, will retrieve these from the storage providers. In addition, they will also retrieve some other random data which will be discarded. Storage agents who don't find any data they have stored in the list will periodically retrieve random data, and send this on as explained below.

The retrieved data is then sent back to the IRC node, but this time via the mix-net. The IRC node then sends each data item back to the C&C node, again via the mix-net. This operation is dependent on the C&C node providing the IRC node with an anonymous return address [26].

Each storage agent[e] responds with its piece of the puzzle, and the IRC node forwards everything to the C&C node:

$$\forall i | IRC \rightarrow C\&C : mix(\{CS_j, return-part, ID_{di}, d_i\})$$

The C&C node then re-assembles the data, and either returns it to the user:

$$C\&C \rightarrow U : \{U, return-full, ID_D, D\}_{K_U}$$

or sends it off to be processed as explained in the next section.

The complete picture is illustrated in Figure 5.

4.5 Processing data in the cloud

When the user wants to do something with the data, it will tell the C&C node:

$$U \rightarrow C\&C : \{process\text{–}cnc, operation, auth, ID_D, N_u\}_{K_{C\&C}}$$

Here, "operation" identifies what should be done, ID_D identifies the dataset, and N_u is a nonce chosen by the user.

The data will first have to be retrieved and re-assembled as explained above. The C&C node then selects an appropriate number of cloud processing providers, depending on the type of data and what is to be done with it. If the data is, e.g., a digital image, and the user wants to manipulate it using a Cloud-based image editor, then the complete data set typically needs to be sent to a single processing provider.

$$C\&C \rightarrow CP_j :$$

$$\{\{CP_j, process-cp, operation, D, K_{PC}, N_c\}_{K_{CPj}}\}_{mix}$$

The data, the nonce chosen by the C&C node, a symmetric key K_{PC} for encrypting the response, and the rest is encrypted with the public key of the cloud processing provider, and sent through the mix-net.

$$CP_j \rightarrow C\&C : mix(\{\{result-cnc, D_{result}, N_c\}_{K_{PC}}\})$$

Note that since the C&C node keeps track of requests to processing providers, the operations are idempotent; replayed responses are ignored, and in case of response failures, a new request will be sent, canceling the former.

The result is returned to the user:

$$C\&C \rightarrow U : \{U, result-user, D_{result}, N_u\}_{K_U}$$

Again, the user will reject any spurious responses with a nonce that doesn't match that of an outstanding request.

If the result is a change in the dataset, it will either have to be re-stored or delivered to the user, depending on the user's wishes. If data items need to be updated or deleted, the authentication mechanism comes into play again. In

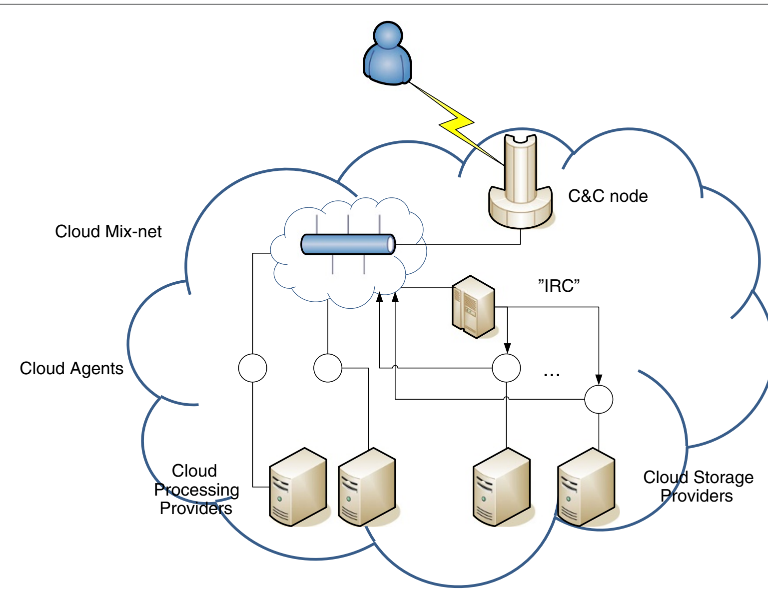


Figure 5 Retrieving and processing information.

any case, a confirmation is sent to the user, closing the outstanding request.

An example of an editing operation is shown in Figure 6. In this case, an image of a rodent (Figure 6a) is to be modified to become a feline (Figure 6d). This example also highlights an optimization opportunity; Figure 6b and 6c identify the modified areas of the image, and on completion only these parts need to be re-stored. The exact mechanisms of how to determine which parts have been changed are beyond the scope of this article, however.

4.6 Implementation considerations

Space does not permit a full implementation specification, but in the following we will illustrate in a little more detail how the actual storage and retrieval process may be realized from the C&C node's point of view.

Although we do not go into specifics here, it is clear that the actual splitting must depend on the type of document. The process is illustrated in Figure 7. A user (or a client running e.g. in a cloud environment) initiates writing of content to the system. The user can configure which storage providers to use for certain file types or content. Part of the config contains information on how each of the storage providers can be used, i.e. description on how to access, write and read content. Typically this can be a proprietary web API. Based on the selection of storage providers available and the content type a recipe is generated. The recipe states the size of blocks the original file is to be split into and a sequence for writing the blocks to the various storage providers. Based on the recipe the content is divided in blocks that each is stored at a storage provider. The recipe is stored, and using the recipe, the content can be retrieved from the storage providers and assembled. The fileID is returned to the initiating part.

The retrieval process is illustrated in Figure 8. A user (or a client running e.g. in a cloud environment) initiates reading of content from the system. The recipe is retrieved based on the fileID. Based on the recipe the file is read from storage providers and assembled. The assembled file is returned to the initiating party.

5 Formal model

We recall from Section 4 that D is a piece of data to be split and stored on a cloud, and *split* is a function that splits D into a sequence H of smaller segments such that $H = (d_1, d_2, ..., d_n)$ where n is the number of segments D shall be split into.

The above process can be denoted as follows.

$$H = split(D) \tag{1}$$
$$= (d_1, d_2, ..., d_n) \tag{2}$$

Recall that H can be represented as $H =< D_s, R_s >$ where

- $D_s = \{d_i | i = 1, ..., n\}$
- $R_s = \{< d_i, d_{i+1} > | i = 1, ..., n-1\}$.

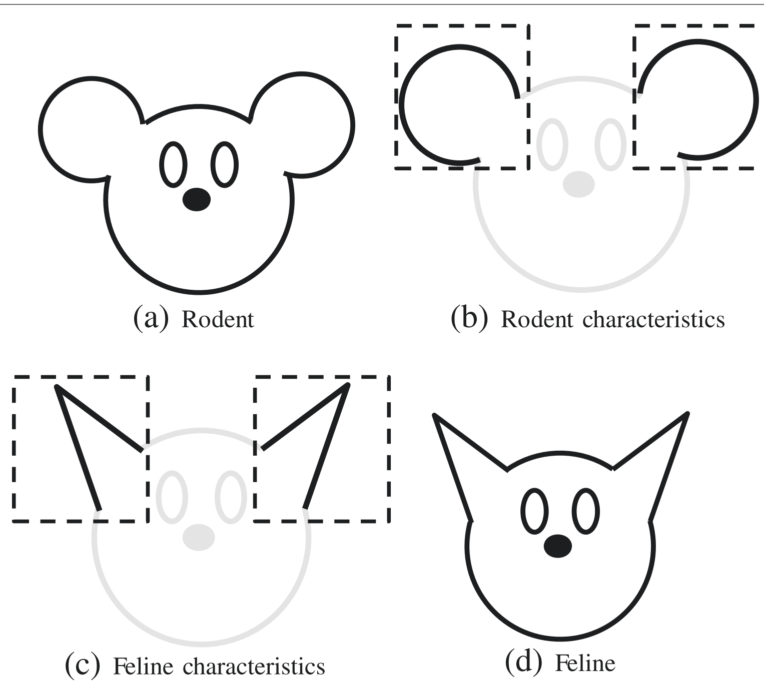


Figure 6 Illustration of editing an image of a rodent to get a feline. (**a**) Rodent. (**b**) Rodent characteristics. (**c**) Feline characteristics. (**d**) Feline.

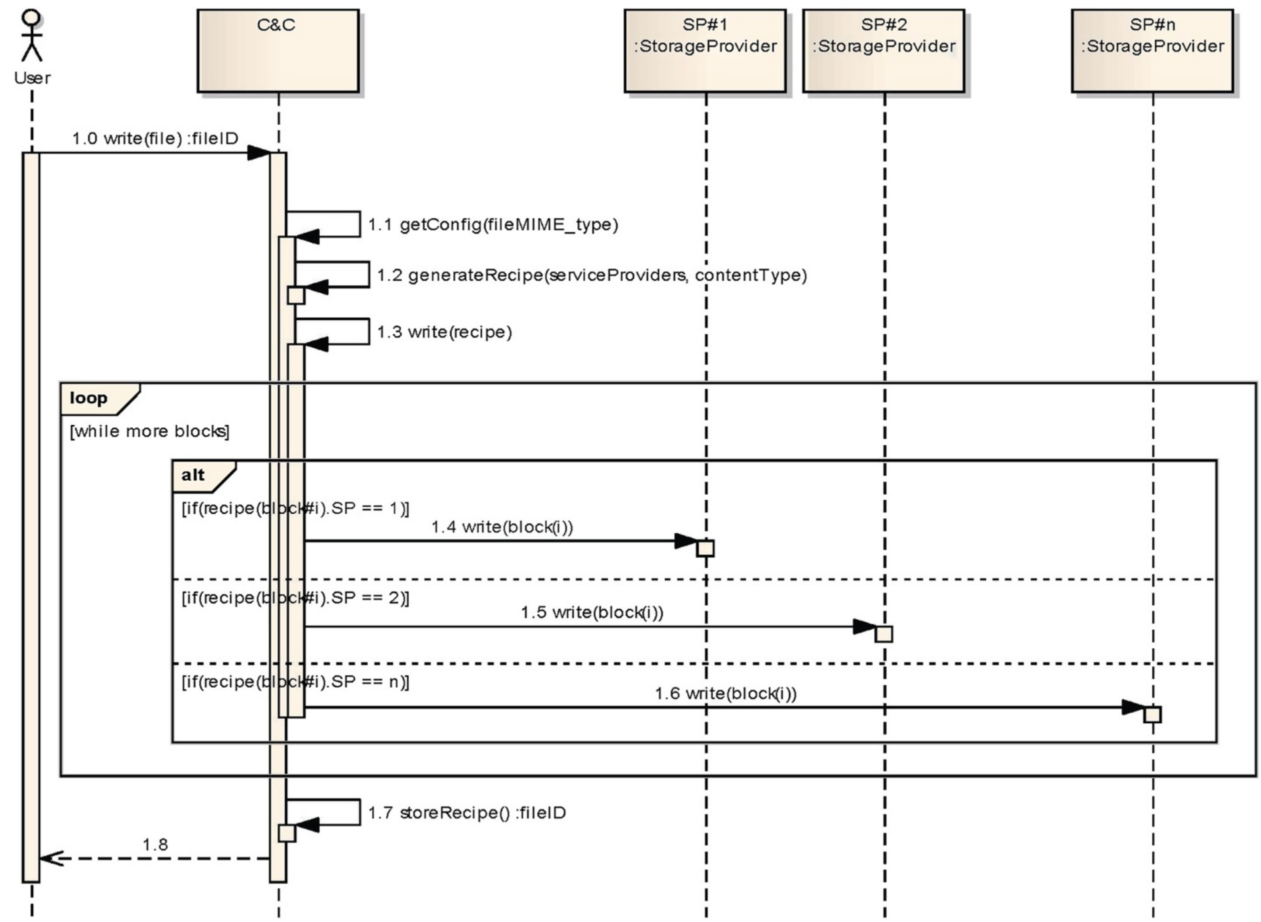


Figure 7 **Sequences for splitting content/file into blocks and writing to a set of storage providers.**

Note that D_s is the set of all segments D is split into. R_s is the set of relations between the segments in D_s, specifying the order of the segments.

The split data is then distributed to different service providers. In general, the dividing of the data is to control the length of each segment to avoid having too much information in a single segment. The permutation of the original data is to diffuse the data in a way that the new presentation conveys very limited information on the original data. Combining the division and permutation could greatly reduce the amount of information carried by a single segment.

5.1 Data segmentation

The criteria stated in section 3 mandate that the data segmentation must make sure that each data segment bears no sensitive information of the original data. The segmentation does not mandate the size of each segment as the size has not direct relationship with the information of the segment bears. Commonly, the bigger a piece of segment, the more information it may bear. Thus a bigger segment is more likely to bear sensitive information.

The criteria do not impose any restriction on the way the original data is segmented. A most simple way to segment data is a sequential segmentation, by which the original data is treated as a binary stream and is divided into multiple substreams in their original order, as illustrated by Figure 9. An alternative way to segment the original data is to pick bytes from random positions of the original data and put them into different segments. Figure 10 shows a case of data segmentation based on the random segmentation approach. The segmentation process is in fact a permutation of the original data, followed by dividing of the permutated data.

The data segmentation is in fact the implementation of the function *split*, where

$$H = split(D) \tag{3}$$

$$= (d_1, d_2, \ldots, d_n) \tag{4}$$

Assuming that e is the maximum amount of information that could be tolerated to be disclosed by any single segment, and *inf* be the function that evaluates the amount of information disclosed.

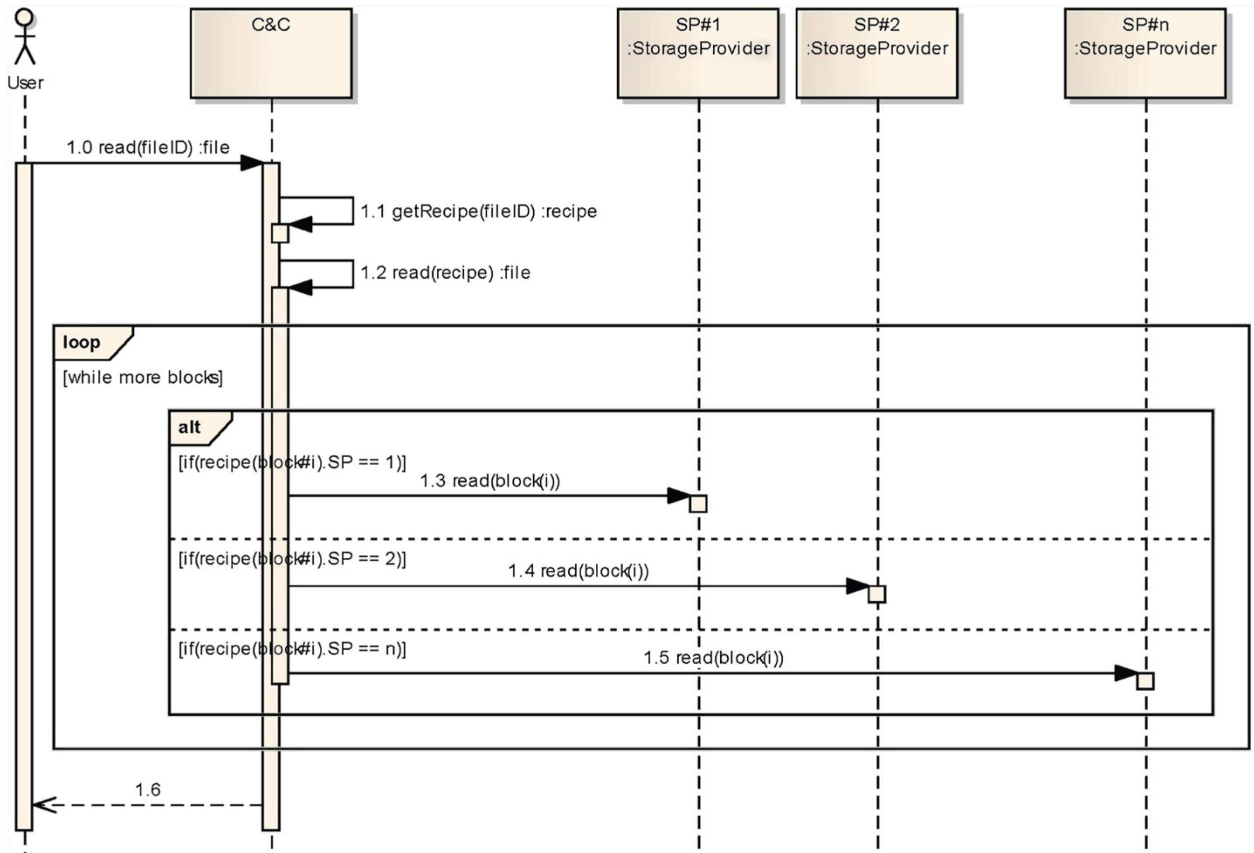


Figure 8 Sequences for retrieving blocks of content from a set of storage providers and assembling them into the original content.

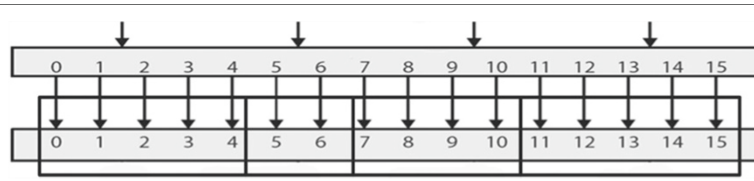


Figure 9 Sequential Segmentation.

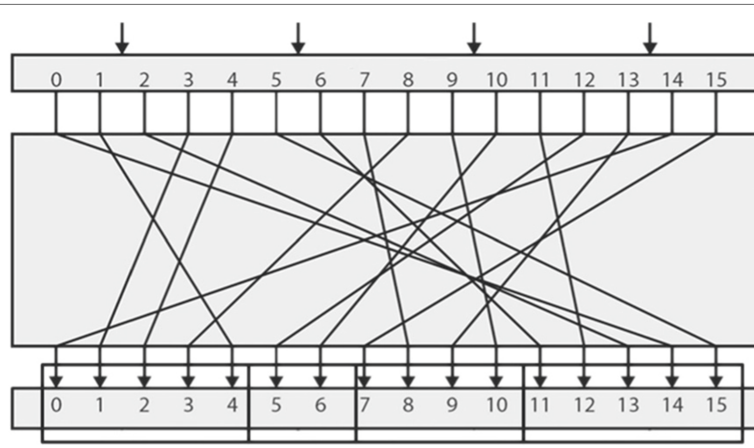


Figure 10 Random Segmentation.

The *split* funciton must make sure that the following holds.

$$\forall d_i \in D, inf(d_i) \leq e \tag{5}$$

Assuming that E is the maximum amount of inforomation that could be tolerated to be disclosed by the whole set of segments, and INF be the function that evaluates the amount of information disclosed by a set of segments, where

$$INF(D_s) = inf(d_1) \times INF(D_s - \{d_1\}) \tag{6}$$
$$= \times_{i=1}^{n} inf(d_i) \tag{7}$$

$\times$ be an operator for calculating the sum of two disclosure degrees.

The *split* funciton must make sure that the following holds.

$$INF(D_s) \leq E$$

5.2 Randomize distribution

Once the original data is transformed into segments, the segments need to be stored on different cloud storage services. This is performed by the segment distribution process, which needs to make sure that the distribution is random to avoid tracking of the segments. Otherwise, segments can be identified by malicious users with limited cost.

Let M be the set of cloud providers that are providing cloud storage services. $\forall m_i \in M$, $\exists D_i \subseteq D_S$ such that $\forall d_j \in D_i$ then d_j is stored in m_i, where

$$\bigcup_{i=1}^{n} D_i = D \tag{8}$$

$$D_j \bigcap D_i = \oslash \tag{9}$$

The distribution process is to generate the set $D_D \subset D_S^*$, where $D_D = \{D_i | i = 1, \ldots, n, D_i$ is the set of segments kept on $m_i\}$. This process can be denoted by a function *dist*, where

$$D_D = dist(D_S, M) \tag{10}$$

Note that D_D is the set of segment sets, specifying the set of segments on each storage server.

An ideal distribution would be a complete random distribution of the segments over the available cloud storage services. A possible way to achieve random distribution is to have the *dist* function randomly pick a cloud storage service for each segment, such that

$$p(d_i \in D_j) \leq \frac{|D_S|}{|M|} \tag{11}$$

where $p(d_i \in D_j)$ denote the possibility that $d_i \in D_j$ holds, $|D_S|$ and $|M|$ represent the number of elements in the set D_S and M respectively.

5.3 Data re-assembling

The re-assembling of the original data requires two pieces of information.

1. The segment distribution information. The segment distribution information allows the picking of related segments from all the cloud storage services. This process is to generate either D_S out from all the segments kept on M without any secret information.
2. The order relations of the segments, R_S. With R_S, the picked data segments, D_S, can be permutated back to the original order to construct the original data.

6 Security analysis

Malicious users can either collect individual data segments or re-assemble the complete data to compromise the data confidentiality.

6.1 Compromization by a single data segment

Malicious users can randomly pick up individual data segments if they have excessive access privilege to the cloud storage services. Any individual data segment that is picked by a malicious user should disclose no information on the original data, according to the criteria of data segmentation. Therefore, it is not possible for malicious users to compromise the confidentiality by any single data segments.

6.2 Compromization by re-assembling

Malicious users can also re-assemble the original data. The re-assembling consists of a few steps as follows.

1. Picking all related data segments.
2. Permuting the data segments.

Picking all the related data segments requires a malicious user to have excessive access privileges to access all the involved cloud storage services, and also requires that the malicious user to pick up all the data segments from all the involved cloud storage services.

Suppose that for each $m_i \in M$, where M is the set of all the cloud storage services, the set of all data segments stored by the cloud storage service m_i is N_i.

The number of data segments stored in M is *TotalSegs* where

$$TotalSegs = \sum_{i-1}^{|M|} |N_i| \tag{12}$$

To be able to re-assemble the original data, a malicious user must be able to pick all the segments and permutate the segments into the right order.

If the malicious user does not know the number of segments the original data has been split into, the total number of possible re-assembled data is *NumOfAllReassembled*, where

$$Num\ Of\ All\ Reassembled = \sum_{i=1}^{U} P^{i}_{TotalSegs} \tag{13}$$

$$= \sum_{i=1}^{U} P^{i}_{\sum_{i-1}^{|M|} |N_i|} \tag{14}$$

Where R is the upper limit of the number of segments that a data is likely to be split into.

If the malicious user knows, s, the number of segments the original data has been split into, the total number of possible re-assembled data is *NumOfReassembled*, where

$$Num\ Of\ Reassembled = \sum_{i=1}^{s} P^{i}_{TotalSegs} \tag{15}$$

$$= \sum_{i=1}^{s} P^{i}_{\sum_{i-1}^{|M|} |N_i|} \tag{16}$$

Both cases require a large amount of computation to brute-force search the complete space. Hence it is not trivial for a malicious user to compromise the data confidentiality by re-assembling the original data.

From a complexity point of view, assuming that there are in total n pieces of data stored in the cloud, let a malicious user try to illegally access a file, which has been split into k pieces and kept in the cloud. The malicious user must first re-assemble the whole file, taking two steps.

1. Step 1: All k pieces must be retrieved corrected out from the n pieces. The probability to retrieve the correct pieces is as follows.

$$p_1 = \frac{1}{C_n^k} = \frac{k!}{n * (n-1) * \cdots * (n-k+1)}$$

2. Step 2: Re-order all the k pieces into the correct order, given the k pieces. The probability of putting all k pieces in the right order without any knowledge of the original data is

$$p_2 = \frac{1}{P_k^k} = \frac{1}{k!}$$

Hence, the probability of re-assembling the file correctly is

$$p = p_1 \times p_2 = \frac{k!}{n * (n-1) * \cdots * (n-k+1)} \times \frac{1}{k!}$$

$$= \frac{1}{n * (n-1) * \cdots * (n-k+1)}$$

Assuming that there is a very large number of pieces in the cloud and each file is split into small enough chunks, n and k are both large enough to ensure that the probability p is small enough to counter attacks.

The cost for an attacker is far from only the computation complexity of re-assembling the k pieces. Due to the distributed characteristics of the proposed storage system, the system contains a very large amount of data and the data are distributed across various networks. The attacker attempting to re-assemble a file by brute force will have to have an extremely large storage space to keep all the retrieved data pieces (both the correct ones and the wrong ones), and it also has to afford the cost for the network bandwidth to transfer such an amount of data across the network.

6.3 Protocol analysis

As can be seen from Table 1, pain has been taken to avoid reusing commands that otherwise might have made it possible to replay messages from A to B as a message from B to C. This is according to Principle 1 of Abadi and Needham [28], and to some extent also Principle 3, since it ensures that every message will only be handled by the same type of actor as intended. However, full adherence with Principle 3 may be difficult to achieve in a setting of anonymous communication. The explicit naming of commands is also in accordance with Principle 10, since it allows for unambiguous encoding of each message.

Since data is sent encrypted from the C&C node to the Cloud Processing provider, it cannot be observed by an

Table 1 Summary of all protocol commands

Command	Explanation
store-full	Command from User to C&C to store a complete dataset
store-part	Command from C&C to IRC to store a piece of data
store-IRC	Command from IRC to storage agent to store a piece of data
retrieve-full	Command from User to C&C to retrieve a complete dataset
retrieve-part	Command from C&C to Cloud Storage provider to retrieve a piece of data
return-part	Cloud Storage provider is returning a piece of data
return-full	C&C node is returning a complete dataset to User
process-cnc	Command from User to C&C to perform processing operation on a dataset
process-cp	Command from C&C to Cloud Processing provider to perform processing operation on a dataset
result-cnc	Cloud Processing provider returning result of processing operation on a dataset to C&C node
result-user	C&C node returning result of processing operation on a dataset to User

adversary, who cannot determine the symmetric key used to encrypt the response, and thus cannot do a suppress-replay attack to replace the result with a bogus result. We are assuming the C&C node has verified public keys to the providers, which means only the selected provider sees the data, but as long as the relationship between user and data is kept secret, it does not really matter exactly *which* Cloud Processing provider handles the data.

By applying the Scyther tool [29], we find (unsurprisingly) that the assumption that data and session keys from the C&C node are kept confidential holds (see Listing 1), but unless the public key of the processing provider has been verified, we cannot assume that it remains confidential. Since the Scyther tool does not support verifying privacy/anonymity claims, it cannot be used to verify the full protocol.

6.4 Listing 1: Scyther code for verifying confidentiality

```
/*
 * Secrecy protocol
 *
/* Uses asymmetric encryption
 */

// PKI infrastructure

const pk: Function;
secret sk: Function;
inversekeys (pk,sk);

// The protocol description

protocol protocol0 (I,R)
{
        role I
        {
                const ni: Nonce;
                const data;
                secret sessionkey: Function;
                send_1(I,R, {I,ni,sessionkey}pk(R) );
                claim_i(I,Secret,ni);
                claim_i(I,Secret,sessionkey);
        }

        role R
        {
                var ni: Nonce;
                var sessionkey: Function;
                read_1(I,R, {I,ni,sessionkey}pk(R) );
        }
}

// An untrusted agent, with leaked information

const Eve: Agent;
untrusted Eve;
compromised sk(Eve);
```

Note that we have made no claims with respect to resource consumption on the cloud providers. Thus, it may be possible for a malfeasor to waste the resources of a Cloud Processing provider by replaying `process-cp` messages from the C&C node. This could be countered by having the Cloud Processing provider store all nonces N_c, and discard all messages with non-fresh nonces, but since this does not contribute to keeping data and users anonymous, it has been omitted to avoid forcing the providers to maintain state information. However, in a possible future commercial solution, this might be solved as part of a payment solution.

7 Simulations

In this section we describe our efforts to simulate the performance of the protocol. The source code and detailed settings are available from the authors upon request.

7.1 Simulation environment

We implemented the protocol utilising a the Nessi simulation framework[30] written in the Python programming language. The main motivation for selecting a Python-based framework was the flexibility and ease of use that the programming language offers despite its obvious performance penalty as compared to other C and C++ based simulation frameworks. The Nessi Framework is not currently actively maintained and lacks a few basic elements such as support for the Internet Protocol (IP), Address Resolution Protocol (ARP) and Transport layer protocols such as TCP and UDP. The framework offer a stack consisting of everything up to and including the Data Link Layer of the OSI reference model as well as some application layer traffic generators. We therefore had to make some simplifying assumptions to cater for the fact that our protocol is designed to run on top of TCP/IP or UDP/IP.

7.2 Simulation implementation

Since our implementation aimed at demonstrating the delay and throughput of the protocol, we simplified the described protocol somewhat. The Mixnet is modelled as a number of nodes connected through a Point-to-Point (P2P) links. Each node contains three network interface cards and is randomly connected to other mixnet nodes through these cards. The Mixnet does not setup a path or route information in the network, instead we specify a *hopcount* that determines the number of times a packet should be forwarded by the network. Forwarding is done by randomly selecting one of the network interface cards attached to the host and then forwarding over the data link layer to the host connected to the other end. When the hopcount reaches zero, the mixnet node forwards the packet to the IRC-node. The IRC-node is connected to the mixnet through an Ethernet bus, and forwards packets to the agents on another Ethernet bus. By utilising the MAC-address as an application level address, our implementation circumvents the problems of not having a network layer protocol. The user and C&C are represented as Traffic generating sources attached to a mixnet-node, whereas the agents are implemented only as Traffic sinks. Hence, the protocol is only implemented in one direction.

7.3 Simulation setup

The default values of all simulation parameters are given in Table 2 and if not explicitly stated otherwise, these are the values used for all simulation runs. We have attempted to keep the values as realistic as possible, without exhausting the resource requirements of the PC running the simulations. This particularly includes memory allocation, which can be quite challenging in Python.

The Mixnet size is given in number of nodes contained in the network and is relatively small compared to what is foreseeable in the Cloud. However, it should be sufficiently large to provide meaningful data. Further, the hopcount, i.e. the number of times a packet is routed inside the mixnet, is set to half the size of the network. The packet size set to 1/10000 of the traffic source data size and the redundancy factor is set to 1. Thus, the default behaviour is not to have redundancy in packet transmission. All of these parameters are configured one at a time for the various simulation runs, except the number of network interfaces per node. This is set to 3, such that each node has a direct connection to three other nodes in the network.

The traffic source is supposed to mimic a kind of FTP-traffic generation. That is, traffic data is characterised by fairly long inter-arrival times and relatively large PDU sizes, which resembles the action of storing data quite well. The simulation time is adapted to the traffic inter-arrival time. The number of agents in the RAIN-network does not really affect the simulation at this point, since we only consider one-way traffic to the agents.

Table 2 Default simulation settings for the RAIN protocol

Mixnet settings	
MIXNET_SIZE*	= 10
MIXNET_HOPCOUNT*	= 5
MIXNET_PACKET_SIZE*	= 1 kB
MIXNET_REDUNANCY_FACTOR*	= 1
MIXNET_LINKS_PER_NODE	= 3
Traffic source settings:	
TRAFFIC_INTERARRIVAL	= 50 s
TRAFFIC_TYPE	= PoissonSource
TRAFFIC_PDU_SIZE*	= 10 MB
SIMULATION_TIME	= 100 s
Network characteristics:	
LINK_DATA_RATE	= 100MB/s
LINK_DISTANCE	= 1000 km
Agent network settings:	
AGENT_NUMBER	= 5

Items marked * are configurable and may vary.

7.4 Results

In this subsection we provide the results and interpretations of our simulation runs related to *delay*, *throughput* and *queue length*.

7.4.1 Delay

Since we have chosen a traffic source that produces burst-traffic, the packet delay as measured on the agents also tend to vary greatly. Figure 11 demonstrates the periodicity of the delay function as compared to the arithmetic mean (dotted line). We measure the delay by recording the time interval between when the packet was sent and when it was received. Hence, we only consider end-to-end delay and do not compute intermediate packet delays (e.g. after k hops).

The packet delay is of course dependent on the distance it has to travel and the amount of time spent being processed and queued along the way. However, in our setup we have fixed the distances between nodes and therefore view delay as a product of changed traffic intensity. In Figure 12 demonstrates that only direct changes to the source traffic either by increasing the PDU size or replicating the packets causes significant changes to the delay. Whereas altering the hop count and packet size of the mixnet have little influence on the end-to-end packet delay. A reason for this may be that additional transmissions within the Mixnet tend to stretch the burst and thereby reduce the congestion problems occurring when to many nodes transmit simultaneously.

7.4.2 Throughput

We measure the throughput as it is seen by the agent, that is, the number of octets received by the agents within the simulation time. The computation is somewhat hampered by the lack of error handling due to missing network and transport layer protocols. However, the average queue

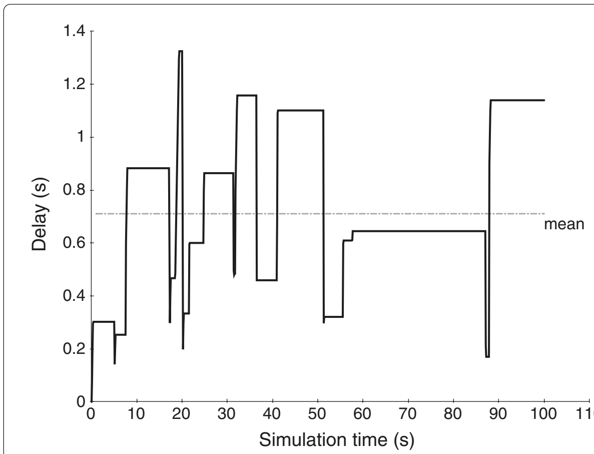
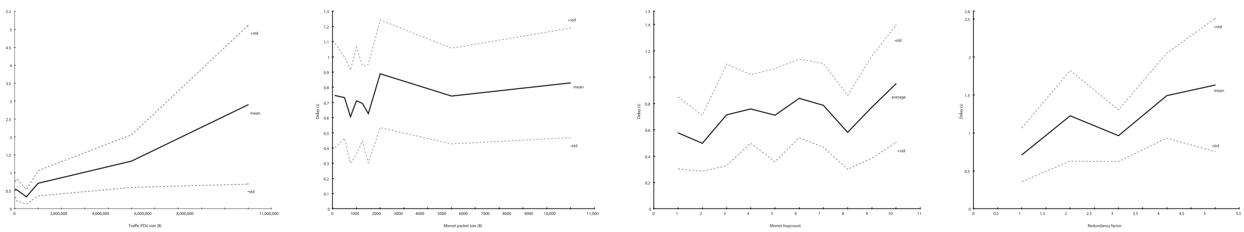


Figure 11 Packet delay on default simulation measured on the agents.



(a) PDU size (b) Packet size (c) Hop count (d) Redundancy

Figure 12 Delay as a function of various factors. (**a**) PDU size. (**b**) Packet size. (**c**) Hop count. (**d**) Redundancy.

time (see next paragraph) suggests that there are not too many packets dropped.

The throughput is dependent on the amount of inbound traffic in the network, the delay and the packet drop rate. Therefore, as can be seen in Figure 13, increasing the traffic through either PDU size, redundancy or number of nodes yields increased throughput. Note however that the increase in network size (number of nodes and traffic sources) is not proportional to the increase in throughput. Doubling the network size yields only about 50% increase in throughput. The hop count does not seemingly affect the throughput consistently. It is unclear why there is a considerable peak around the default hop count value (MIXNET_HOPCOUNT= 5).

7.4.3 Queue length

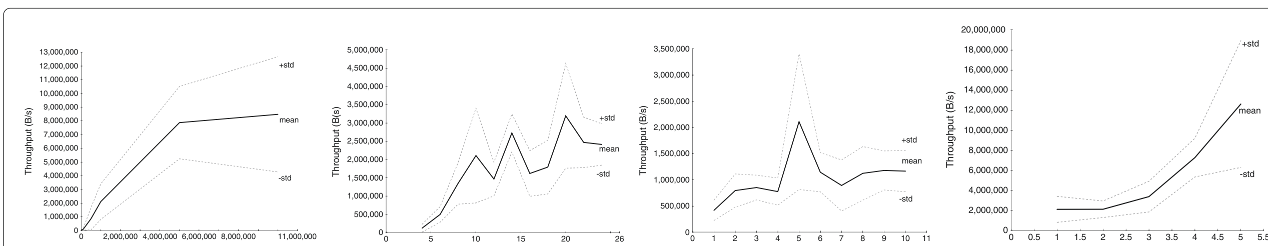
We measure the queue length on all nodes in the Mixnet at with a sampling frequency of 10/s. Any changes to the queue between these samples are not detected and hence we only provide an estimate of the queue length. Note that we measure the total queue length, i.e. the sum, on all nodes in the Mixnet. The packet delay described above is partially dependent on the queue length, such that the longer the queue, the longer the delay. However, the reason for measuring the queue length of Mixnet nodes is that are indicators for network congestion.

From Figure 14 we see that both the size of traffic source generated PDUs and the redundancy factor greatly influence the queue length. Also, the Mixnet packet size affects the queue length as larger sizes reduces the queues to a level close to zero. The hopcount has only moderate effect on the queue, which is in line with what we discovered for packet delay and throughput. Note however, that although the average total queue length is quite small, the standard deviation is considerable, indicating that the queue may be considerable for shorter periods of time. This is expected since the traffic source we have selected, generates packets in bursts which yields a burst behaviour in the queues as well.

8 Discussion

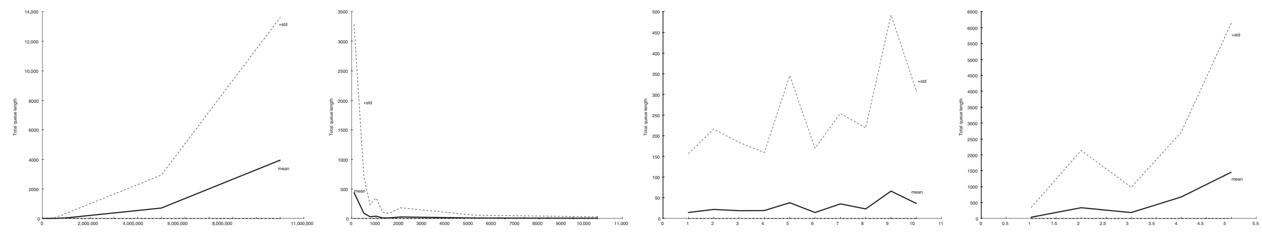
Cloud service providers have been identified as potential targets of attack simply because of the vast amounts of data they store on behalf of their multitude of customers. In this sense, it may be in the providers' best interest to "know less" - if even the provider cannot access the customers' data directly, there is little point in attacking them.

Strictly speaking, most users would probably be happy if it were possible to impose universal usage control [31] on data submitted to providers (a sort of "reverse DRM", where end-users get to control how multi-national corporations use their data), but despite Krautheim's efforts [12], we do not believe this will be a reality in the foreseeable future. Thus, it would seem that the easiest way to control what a provider does with your information is to hide it - either through encryption (as previously proposed for the storage providers) or through separation. A brief comparison of RAIN with other such approaches is provided in Table 3.



(a) PDU size (b) # nodes (c) Hop count (d) Redundancy

Figure 13 Throughput as a function of various factors. (**a**) PDU size. (**b**) # nodes. (**c**) Hop count. (**d**) Redundancy.



(a) PDU size (b) Packet size (c) Hop count (d) Redundancy

Figure 14 Queue size as a function of various factors. (**a**) PDU size. (**b**) Packet size. (**c**) Hop count. (**d**) Redundancy.

In a real-life setting, there will be cases where very small units of data carry a significant amount of sensitive information, such as blood type for patients. It will thus be imperative that not only shall it not be possible to match e.g. a blood type to an identity, but in storage it should also not be possible to determine what the data item refers to.

In this respect, our approach is different from the *k*-anonymity concept proposed by Sweeney [39], since we assume that it generally will *not* be possible for an observer to deduce what the data mean; Sweeney's concept is meant to ensure "statistical anonymity" by ensuring that at least *k* different persons have the same characteristics in any given dataset.

8.1 Searching and indexing

One major unsolved problem with our solution is related to searching and indexing. Even if it were possible to create an index to search in, where can we store it? Thus, we currently have to accept that searching is not possible without re-constructing each file first.

8.2 Business model

It's been said that everybody wants security, but nobody is willing to pay for it. This means that not only is it difficult to get funding for security measures in organizations where security is viewed as a net expense, but most users are also not willing to put up with the extra inconvenience that added security mechanisms often imply. Contrasting this with the (at least currently) free services such as Google Docs [40] that cloud providers are throwing at customers, it may be hard to imagine anybody paying money to get the same thing "more secure".

However, privacy is evidently an issue for some people, as the usage statistics of the TOR network can testify [19], and also experiments in Scandinavia have shown that many people will choose privacy if it's made available to them [41]. In general, it is dangerous to confuse the concepts of "privacy" and "confidentiality", but in our case we believe that the privacy aspects will be the major driver for people wanting to keep their data confidential.

Table 3 Comparison of RAIN with other approaches of splitting data

Approach	Summary	Misgivings
Singh et al. [32]	A scheme for *n-out-of-m* secret sharing of data [33]	Do not provide an algorithm for the actual splitting of the data to be stored.
Parakh and Kak [34]	Another *n-out-of-m* scheme	Do not discuss why their scheme should be better than e.g. the one proposed by Rabin [35].
Luna et al. [36]	Yet another *n-out-of-m* scheme, but add an additional concept of *Quality of Security* (QoSec) to rate individual storage providers.	Solution is tailored to a Grid computing scenario, not to commercial cloud operators.
RACS [37]	Prevents vendor lock-in and data loss through failures by performing striping of data (in RAID-5 fashion) across multiple cloud providers	Does not offer privacy or confidentiality.
Mnemosyne [38]	Offers steganographic storage which not only hides data, but also prevents anyone from determining that there is anything hidden in the first place.	Mnemosyne encrypts each block, and thus requires a key management system in addition to the information dispersal algorithm.
Free Haven [13]	A collaborative distributed storage system, based on peer-to-peer file sharing principles.	Does not provide a new solution for the anonymous communications channel, but uses a set of anonymous remailers as a basis.
OceanStore [14]	Provides distributed storage	Is not concerned with ensuring anonymity of the individual users.
ShareMind [16]	Offers distributed privacy-preserving computations	Is not focused on storage, only computation.

How to pay for the services anonymously has not been completely resolved. Most existing solutions such as TOR [19] and Free Haven [13] are based on volunteer or barter arrangements, where participants get free use of the service by supporting parts of it on their own systems. The payment problem is also the main obstacle for Chaum's approach [42], since he assumes the existence of a digital cash system, something which remains elusive. Still, since the cloud paradigm is oriented toward *pay per use*, we believe it will be easier to solve this in the clouds than in many other situations.

8.3 Trust

Since we place all our trust in the C&C node, it will remain as a "single point of trust" as long as it is realized as part of the cloud. It would have been desirable to strengthen this by ensuring that the C&C node provider only sees the information as we see it ourselves, preventing it from mining stored information. However, as long as the C&C node is required to keep track of all the data items (or slices, as in the example), there is nothing to prevent it from accessing this information as it pleases. Currently, only the alternative solution in Figure 2b keeps this information out of the cloud.

However, we maintain that even if we still have to trust the C&C node in the cloud, this in an improvement over handing all our data over to Google. The C&C node in effect plays the role of a Trusted Third Party, and generally does not need to have the enormous resources of the current commercial cloud providers. Thus, the C&C node could in principle be run by some small company in the user's neighbourhood, enabling a traditional trust relationship.

9 Further work

This divide-and-conquer approach may be suitable for privacy-conscious home users and small businesses, but the ultimate holy grail is absolute confidentiality in the cloud, and thus a deliverance from trust. Only then can cloud computing deliver on the dream of computing power as a utility akin to power, water and gas. A further refinement of our approach that removes the necessity to trust the C&C node is therefore a natural challenge.

We have implemented a simple proof-of-concept prototype [43], so the next step will be to implement a large-scale prototype to gauge performance impacts on typical cloud applications. One particular challenge in this respect is to determine the optimal slicing strategy for arbitrary data. It is likely that a trade-off between security and efficiency will have to be made in order to capitalize on the advantages of the Cloud Computing paradigm. The prototype will be targeted toward a "sensitive but unclassified" application, representing a realistic use case.

10 Conclusion

We have described the design of the Redundant Array of Independent Net-storages (RAIN) that achieves confidentiality in the cloud through dividing data in sufficiently small chunks. We have provided a formal model and security analysis to motivate our claims, and our simulation results indicate that the efficiency of our approach is acceptable. We believe that this may be useful for small office and home users, but experimentation and practical experience will be necessary to validate the approach.

11 Endnotes

[a]It may be a matter for debate whether the solution rather should have been referred to as *confidentiality*-preserving.
[b]This is a rather fuzzy assumption, but it is clear that if there are only two parties communicating, an adversary can trivially determine that all data observed leaving one party will arrive at the second party. This assumption of "more than a few" users is also used in, e.g., TOR [19].
[c]In the current description, we make no attempt to hide the identity of the user from the C&C node, so here we could assume that the authentication is an (as yet unspecified) conventional mechanism authenticating the user to the C&C provider. However, in the following we will use this authentication mechanism to control access to individual data items, and in this case the cloud providers should of course not know the identity of the user or other actors.
[d] The number of bogus IDs requested is configurable.
[e]It may be debatable whether it makes sense to include the provider ID in the response, but certainly the C&C node ID cannot be there, as it is supposed to be anonymous.

Competing interests

The authors declare that they have no competing interests.

Acknowledgements

This work has been supported by Telenor through the SINTEF-Telenor research agreement, by China State Key Lab of Software Engineering through grant SKLSE2010-08-22, and by China Canton-HK research project TC10-BH07-1.

Author details

[1]SINTEF ICT, Trondheim, Norway. [2]South China Normal University, Guangzhou, China. [3]University of Western Macedonia, Florina, Greece. [4]Telenor Research and Future Studies, Trondheim, Norway.

Author's contributions

MGJ initated the RAIN concept, constructed the RAIN protocol and drafted the article. GZ constructed the formal model and performed the security analysis. AVV contributed to the formal model. ÅAN performed the simulations and wrote up the results. SA provided the implementation considerations. YT contributed to the security analysis. All authors read and approved the final manuscript.

Received: 30 January 2012 Accepted: 1 June 2012
Published: 17 July 2012

References

1. Zhao G, Rong C, Jaatun MG, Sandnes F (2012) Reference deployment models for eliminating user concerns on cloud security. J Supercomputing 61(2): 337–352. http://dx.doi.org/10.1007/s11227-010-0460-9 [10.1007/s11227-010-0460-9]

2. Chen Y, Paxson V, Katz RH (2010) What's new about cloud computing security? Technical Report UCB/EECS-2010-5, EECS Department, University of California, Berkeley. http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-5.html
3. Nyre ÅA, Jaatun MG (2010) A probabilistic approach to information control. J Internet Technol 11(3): 407–416
4. European Parliament (1995) Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data
5. Hinke TH (1988) Inference aggregation detection in database management systems. In: Proceedings of the 1988 IEEE conference on, Security and privacy, SP'88. IEEE Computer Society, Washington, DC, USA, pp 96–106. http://portal.acm.org/citation.cfm?id=1949221.1949237
6. Jaatun MG, Nyre ÅA, Alapnes S, Zhao G (2011) A Farewell to, Trust: An Approach to Confidentiality Control in the Cloud. In: Proceedings of the 2nd International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless Vitae Chennai 2011). IEEE, Piscataway, NJ
7. Zhao G, Rong C, Jaatun MG, Sandnes F (2010) Deployment Models: Towards Eliminating Security Concerns from Cloud Computing. In: Proceedings of the International Conference on High Performance Computing & Simulation. pp 189–195
8. Jensen M, Schwenk J, Gruschka N, Iacono LL (2009) On Technical Security Issues in Cloud Computing. In: Cloud Computing, IEEE International Conference on, Volume 0. IEEE Computer Society, Los Alamitos, pp 109–116
9. Whitney L (2009) Amazon EC2 cloud service hit by botnet, outage. http://news.cnet.com/8301-1009_3-10413951-83.html
10. Ristenpart T, Tromer E, Shacham H, Savage S (2009) Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In: Proceedings of the 16th ACM conference on Computer and communications security. ACM New York, pp 199–212
11. Chen A (2010) GCreep: Google Engineer Stalked Teens, Spied on Chats. Http://gawker.com/#!5637234/gcreep-google-engineer-stalked-teens-spied-on-chats
12. Krautheim F (2009) Private virtual infrastructure for cloud computing. In: proceedings of the Workshop on Hot Topics in Cloud Computing, HotCloud
13. Dingledine R, Freedman MJ, Molnar D (2000) The Free Haven Project: Distributed Anonymous Storage Service. In: Proceedings of the Workshop on Design Issues in Anonymity and Unobservability
14. Rhea S, Eaton P, Geels D, Weatherspoon H, Zhao B, Kubiatowicz J (2003) Pond: the OceanStore Prototype. In: Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03)
15. Bogdanov D, Laur S, Willemson J (2008) Sharemind: a framework for fast privacy-preserving computations. Cryptology ePrint Archive, Report 2008/289. Http://eprint.iacr.org/
16. Cybernetica News blog - Sharemind (2008). http://research.cyber.ee/sharemind/. Http://research.cyber.ee/sharemind/, visited: Sept. 9, 2010
17. Wang P, Wu L, Aslam B, Zou CC (2009) A Systematic Study on Peer-to-Peer Botnets. In: ICCCN '09: Proceedings of the 2009 Proceedings of 18th International Conference on Computer Communications and Networks. IEEE Computer Society, Washington, pp 1–8
18. Zhao G, Jaatun MG, Vasilakos A, Nyre ÅA, Alapnes S, Ye Q, Tang Y (2011) Deliverance from Trust through a Redundant Array of Independent Net-storages in Cloud Computing. In: Proceedings of IEEE Infocom
19. Dingledine R, Mathewson N, Syverson P (2004) Tor: The second-generation onion router. In: Proceedings of the 13th conference on USENIX Security Symposium-Volume 13. USENIX Association, Cerrito, pp 21–21
20. Wooldridge M (2002) An Introduction to MultiAgent Systems. John Wiley & Sons Ltd, Chichester
21. Li X, Zhang H, Zhang Y (2009) Deploying Mobile, Computation in Cloud Service. In: Jaatun M, Zhao G, Rong C (eds) Cloud Computing, Volume 5931 of Lecture Notes in Computer Science. Springer, Berlin / Heidelberg, pp 301–311. http://dx.doi.org/10.1007/978-3-642-10665-1_27. [10.1007/978-3-642-10665-1_27]
22. Zhang Z, Zhang X (2009) Realization of open cloud computing federation based on mobile agent. In: Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on, Volume 3. pp 642–646
23. Aversa R, Di Martino B, Rak M, Venticinque S (2010) Cloud Agency: A Mobile Agent Based Cloud System. In: Complex, Intelligent and Software Intensive Systems (CISIS), 2010 International Conference on. pp 132–137
24. Borselius N (2002) Mobile agent security. Electron & Commun Eng J 14(5): 211–218
25. Jaatun MG, Zhao G, Alapnes S (2011) A Cryptographic Protocol for Communication in a Redundant Array of Independent Net-storages. In: Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2011)
26. Chaum DL (1981) Untraceable electronic mail, return addresses, and digital pseudonyms. Commun ACM 24: 84–90. http://doi.acm.org/10.1145/358549.358563
27. McCoy D, Bauer K, Grunwald D, Kohno T, Sicker D (2008) Shining Light in, Dark Places: Understanding the Tor Network. In: Borisov N, Goldberg I (eds) Privacy Enhancing Technologies, Volume 5134 of Lecture Notes in Computer Science. Springer, Berlin / Heidelberg, pp 63–76. http://dx.doi.org/10.1007/978-3-540-70630-4_5
28. Abadi M, Needham R (1996) Prudent engineering practice for cryptographic protocols. Software Eng, IEEE Trans 22: 6–15
29. Cremers C (2006) Scyther - Semantics and Verification of Security Protocols. Ph.D. dissertation, Eindhoven University of Technology
30. Vernez J, Ehrensberger J, Robert S (2006) Nessi: A Python Network Simulator for Fast Protocol Development. In:Computer-Aided Modeling, Analysis and Design of Communication Links and Networks, 2006 11th International, Workshop on. IEEE, Piscataway, NJ, pp 67–71
31. Park J, Sandhu R (2004) The UCON_ABC usage control model. ACM Trans Inf Syst Secur 7: 128–174
32. Singh Y, Kandah F, Zhang W (2011) A secured cost-effective multi-cloud storage in cloud computing. In: Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on. pp 619–624
33. Shamir A (1979) How to share a secret. Commun ACM 22: 612–613. http://doi.acm.org/10.1145/359168.359176
34. Parakh A, Kak S (2009) Online data storage using implicit security. Inf Sci 179(19): 3323–3331. http://www.sciencedirect.com/science/article/pii/S0020025509002308
35. Rabin MO (1989) Efficient dispersal of information for security, load balancing, and fault tolerance. J ACM 36(2): 335–348
36. Luna J, Flouris M, Marazakis M, Bilas A (2008) Providing security to the Desktop Data Grid. In: Parallel and Distributed Processing 2008 IPDPS 2008 IEEE International Symposium on. pp 1–8. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4536443
37. Abu-Libdeh H, Princehouse L, Weatherspoon H (2010) RACS: a case for cloud storage diversity. In: Proceedings of the 1st ACM symposium on Cloud computing, SoCC '10. ACM, New York, NY, USA, pp 229–240. http://doi.acm.org/10.1145/1807128.1807165
38. Hand S, Roscoe T (2002) Mnemosyne: Peer-to-Peer Steganographic Storage. In: Revised Papers from the First International Workshop on Peer-to-Peer Systems, IPTPS '01. Springer-Verlag, London, pp 130–140. http://dl.acm.org/citation.cfm?id=646334.756802
39. Sweeney L (2002) k-anonymity: A model for protecting privacy. Int J Uncertainty, Fuzziness and Knowledge-Based Syst 10(5): 557–570
40. Google (2011) Google Docs - Online documents, spreadsheets, presentations, surveys, file storage and more. Http://docs.google.com
41. Larsen NE (2009) Privacy in The Polippix Project. In: D 7.3 PRISE Conference Proceedings: "Towards privacy enhancing security technologies - the next steps". pp 143–149
42. Chaum D (1988) The dining cryptographers problem: Unconditional sender and recipient untraceability. J Cryptology 1: 65–75. http://dx.doi.org/10.1007/BF00206326 [10.1007/BF00206326]
43. Jaatun MG, Askeland C, Salvesen AE (2012) Drizzle: The RAIN Prototype. In: "Proceedings of the 12th International Conference on Innovative Internet Community Systems"

doi:10.1186/2192-113X-1-13
Cite this article as: Jaatun *et al.*: **The design of a redundant array of independent net-storages for improved confidentiality in cloud computing.** *Journal of Cloud Computing: Advances, Systems and Applications* 2012 **1**:13.